2006

A Multilevel Secure Relational Database Model with key-polyinstantiation

Natalia Stakhanova
Iowa State University

Ruchi Dhingra
Iowa State University

A Multilevel Secure Relational Database Model with key-polyinstantiation

Abstract
The problem of recognizing motifs from biological data has been well-studied and numerous algorithms, both exact and approximate, have been proposed to address the underlying issue. We strongly believe that open availability and ease of accessibility of quality implementations for such algorithms are critical to the research community, in order to directly reproduce and utilize the results from other studies, so as not to reinvent the wheel. Moreover, it is also important for the implementation to be as generic as possible so that any researcher can to extend it with minimal effort to test a newly implemented algorithmic extension or heuristic. With this motivation, we choose to focus an existing algorithm, PatternBranching and, to a lesser degree, Yang2004. We analyze these approaches for minor heuristical changes & speed-ups by adjusting certain thresholds, and finally, implement the variant in high-level language (Java) using thought through programming practices and generic, extensible interfaces. We also analyze the performance of PatternBranching using a synthetically generated test-suite for a variety of sequence lengths and report the results. Code from this project will be made freely available online to the research community.

Disciplines
Computer Sciences

A Multilevel Secure Relational Database Model with key-polyinstantiation (MLS-K)

Natalia Stakhanova Ruchi Dhingra

Department of Computer Science
Iowa State University
Ames, Iowa

Abstract

In multilevel security there is a hierarchy of users or user-levels, in which each user has its own version of information. Most of the existing multilevel secure (MLS) data models support u-polyinstantiation. The only model that supports key-polyinstantiation was proposed by Gadia et al[GS1998, JS1990, CG1995], but work on it remains incomplete. It is important for a model to support key-polyinstantiation because in the real world it is often the case that an object varies in its key value(s) (such as name, SSN, identification number etc.) when it occurs in the beliefs of different users. Thus having a unique key across beliefs limits our ability to accurately model the real world. Our work focuses on the relational database model, supports key-polyinstantiation and has semantics defined in an SQL-like format since most database users are experienced in using SQL and hence such semantics are intuitive and easy to understand.

1. Introduction:

In multilevel security there is a hierarchy of users or user-levels[1], in which each user has its own version of information. A user can see all the information belonging to itself and to users below his level. On the other hand, information belonging to a higher user, or even existence of such information or such user-levels, is hidden from lower user-levels. A model for a multilevel security database must be devoid of covert channels that can compromise of user confidentiality.

Most of the existing multilevel secure (MLS) data models support u-polyinstantiation[2] [JS1990, JS1991, PMP1994, SC1998, SW1992, WSQ1994]. These have been defined for relational databases and have semantics very similar to SQL. They also have the potential to be implemented in SQL, although no such implementation appears to have been carried out[3].

The only model that supports key-polyinstantiation[4] was proposed by [CG1995, GS1998]. This model also supports the relational model, but work on it remains incomplete – no operational semantics or implementation have been specified.

It is important for a model to support key-polyinstantiation because in the real world it is often the case [GS1998] that an object varies in its key value (s) (such as name, SSN, identification number etc.)

[1] We use both terms user and user-level interchangeably in this paper when the meaning is apparent from the context.
[2] Under u-polyinstantiation it is assumed that a real world object has the same key under beliefs of all users that can access the object, although non-key values may vary.
[3] SeaView has an implementation but we did not review since these works are advancements on SeaView.
[4] Key-polyinstantiation allows key as well as non-key attributes to vary across user beliefs.

when it occurs in the beliefs of different users. Thus having a unique key across beliefs limits our ability to accurately model the real world.

Our work focuses on the relational database model and we hope to extend it to other database types (spatial, temporal etc.) over time.

We also see merit in having operational semantics defined in an SQL-like format since most database users are experienced in using SQL and hence such semantics are intuitive and easy to understand.

These factors motivated this research effort and our objective was – **'To define a model for a multi-level secure relational database that supports key polyinstantiation and whose statements have semantics closely resembling SQL'.**

Our model – we call it MLS-K- supports various integrity constraints and we believe that our SQL-like statements can be easily implemented using SQL constructs[5].

In the following section we summarize exiting research and highlight some of the issues that current models fail to address. In section 3, we define our MLS-K model – its semantics, interpretation, satisfiability of key-polyinstantiation and its integrity constraints. This is followed by the syntax and semantics of the four basic SOL-like queries – SELECT, INSERT, UPDATE and DELETE. In section 4 we briefly comment on the model evaluation. Then we conclude with a summary of our model and areas for future work and cite references.

2. Related Work:

Most of the existing MLS data models support only u-polyinstantiation [JS1990, JS1991, PMP1994, SC1998, SW1992, WSQ1994]. These have been defined for relational databases and have semantics similar to SQL. They support various integrity constraints and ensure enforcement of the 'read-down and write up' policy. They also have the potential to be implemented in SQL, although no such implementation appears to have been carried out.

The only known model that supports key-polyinstantiation was proposed by [CG1995, GS1998]. This model was defined for temporal and spatial databases but it also supports the relational model and it was the first work to introduce the concept of anchors. But work on this model largely remains incomplete – no operational semantics or implementation has been specified.

3. Proposed Work:

In a multilevel security environment, the user-levels form a hierarchy[6] where every user has its own object space. We assume that each object is uniquely identified by its key values in a given object space; although an object can have different keys when viewed in the object space of different users.

Information available to a user consists of the following:

1. The object space which is the user's belief about which objects exist in the real world.

[5] Implementation not included in the scope of this work.
[6] We use the alongside drawn user-level hierarchy in all our examples.

2. Property values of objects in the user's object space.
3. Knowledge of which object in user's object space is known to a lower user, possibly with different identity (key) and attribute values.

As an example, μ believes in the existence of 2 objects – Mary and John, α believes in the existence of 2 objects – Inga and Tom and λ believes in the existence of 2 objects – Hari and Ron as shown in Figure 1.

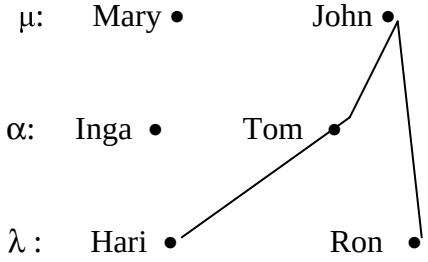


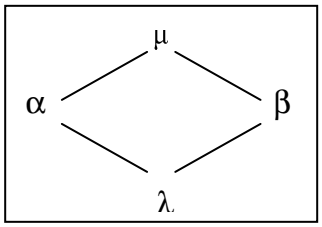
Figure 1.

Now α thinks that Tom and Hari are the same object and Ron does not exists in the real world. Although μ believes that John is known to λ as Ron and an object Hari does not exist.

The object space of each user is independent and identical in its schema[7]. And an upper user has only read access to lower users. To maintain confidentiality of information, it is important for the upper user to not have a write access to a lower user, and for a lower user to not have a read access to the upper user. Therefore, a higher user has only write access to its own belief data.

In a model for secure databases such communication cannot be also created covertly by users through the facilities available in the model. In other words the model should avoid *covert channel*s.

The same example shown in Figure 1 may be represented in tuples in the following way:



[7] Thus, if a higher user wishes to add attributes to a relation schema that should not be visible to the lower level users, (s)he should add a new relation.

what user sees		beliefs		
μ	John	μ	John	
μ	John	α	Tom	
μ	John	λ	Ron	
μ	Mary	μ	Mary	
α	Tom	α	Tom	Other attributes
α	Tom	λ	Hari	
α	Inga	α	Inga	
λ	Hari	λ	Hari	
λ	Ron	λ	Ron	

Figure 2

Then the first two columns represent a user who sees this particular tuple and the attributes by which he identify the corresponding object. The next two columns represent the user whose object is believed by the first user. For example, user μ believes α that α:Tom is the same object as μ:John.

3.1 Semantics:

Below we capture these intuitive ideas as formal semantics of the MLS-K data model.

1. The structure of the model can be represented in a form of the security hierarchy of users where user-levels are partially ordered in lattice[8].
2. Each user is assigned a *security classification* (or classification) which defines what data is visible to this user[9].
3. The relation schema is: R(OC,OO, BC,BO, A_1, C_1, A_2, C_2, …, A_k, C_k) where
   OC- owner classification is the security classification of the tuple owner.
   OO- owner object is the object key (could be multiple attributes) as believed by the tuple owner.
   BC- belief classification is the security classification of the relation owner.
   BO-belief object is the object key (could be multiple attributes) as believed by the relation owner.
   A_i –data attribute over domain D_i
   Ci –classification attribute for A_i. The domain of C_i is specified by a set $\{L_i..H_i\}$ containing all security classifications ranging from L_i up to H_i ($L_i < H_i \leq$ OC).
4. Let r denote an instance of a multilevel relation R.

[8] A lattice is a partial order defined on the user-levels. In our example, the partial order is μ > α, μ > β, α > λ, β > λ, and under transitive closure of this partial ordering μ > λ. So we say μ is higher than α, α is higher than λ and β is higher than λ. Further, under transitive closure of the partial ordering, μ is higher than λ.

[9] We use the user-level as the security classification although this is not mandatory.

Then in Figure 2 columns should be labeled as follows:

OC	OO	BC	BO	A_i	Ci	…
μ	John	μ	John			
μ	John	α	Tom			
μ	John	λ	Ron			
μ	Mary	μ	Mary			
α	Tom	α	Tom	Other	attributes	
α	Tom	λ	Hari			
α	Inga	α	Inga			
λ	Hari	λ	Hari			
λ	Ron	λ	Ron			

Figure 3

3.2 Interpretation:

1. Objects *are believed* by δ-user[10] to exist if the following equality of tuples holds, t[OC,OO] = t[BC,BO].
2. Objects *are accepted* if t[OC,OO] ≠ t[BC,BO]. We say δ-user believes that an object OO is known to BC-user as object BO. And we say that the data (attributes) of object BO is accepted by δ-user.
3. Objects are *not accepted* by a user if they are neither believed nor accepted.

OC	OO	BC	BO
α	Tom	α	Tom
α	Tom	λ	Hari
λ	Hari	λ	Hari

Figure 4

For example, α believes that there is an object Tom in the real world, this is reflected in the first tuple, and this object Tom is known to λ as Hari.

A user's belief comprises of all objects that have been believed and accepted by the user.

3.3 Key-polyinstantiation:

For a given relation, the set of (OC,OO,BC,BO) values is unique and said to be a key of a relation. Two or more tuples could have the same BC and BO values (i.e. they are believed by the same user) or the same OC and OO values (i.e. they are owned by the same user) but not both.

[10] Note that δ is not a user in our hierarchy and when we use this notation we mean that this property is valid for the current user who could be at any level in the hierarchy i.e. δ can be μ, α, β or λ as needed.

3.4 Integrity Properties:

1. **Entity Integrity**:
   a) t[OC] ≠ null and t[BC] ≠ null and t[BO] ≠ null i.e. no attribute of the key can be null except OO.
   b) If t[OO] contains more than one data attribute then all of them have the same classification t[OC][11].
   c) If t[BO] contains more than one data attribute then all of them have the same t[BC].
   d) Any other attribute A_i or C_i can take any value from its respective domain or it can take a null value.
2. **Null Values**:
   a) t[OO] = null means that the tuple was inserted by lower user and the current user is expected to accept or reject these data. For example,

μ: null	α: Aaron
α: Aaron	α: Aaron

Figure 5

If user α introduced a new object 'Aaron', since μ > α the data was automatically inserted to μ level. However, μ has not yet decided whether to believe α:Aaron or not.

   b) $t[A_i]$ = null means that the user has not inserted a value for this particular attribute and $t[C_i]$ = null means that the tuple was deleted by the owner of this value, so there existed $t' \in r$, $t'[OC] = t[C_i]$. This is of significance only for borrowed data.

3. **Data Borrow Integrity**: An instance r of a multilevel relation R satisfies the data-borrow integrity iff $\forall t \in r$ and $1 \leq i \leq n$,
   a) If $\forall C_i$ $t[C_i] \leq t[BC]$, which means that all data in particular relation must either belong to the δ-user or be borrowed from the lower level users.
   b) If $t[BC,BO] \neq t[OC,OO]$ then $\exists t' \in r$ such that $t[BC,BO] = t'[OC,OO] = t'[BC,BO]$ and $\forall A_i, C_i$ $t[A_i, C_i] = t'[A_i, C_i]$. In other words if an object was accepted by δ-user then this object must exist on some lower level and must be believed by the lower level user.
   c) If $t[A_i] \neq null \wedge t[C_i] < t[BC] \leq t[OC]$, then $\exists t' \in r$ such that $t[BC,BO] = t'[OC,OO] \wedge t[C_i] = t'[BC] \wedge t[A_i, C_i] = t'[A_i, C_i]$. A user cannot borrow attributes from the object whose existence it does not accept.

[11] We have kept OO and BO as single attributes in our examples for ease of understanding but this is not mandatory.

OC	OO	BC	BO	Age	C_{age}	Salary	C_{salary}
α	Tom	α	Tom	35	α	80	λ
α	Tom	λ	Hari	39	λ	80	λ
λ	Ron	λ	Ron	40	λ	54	λ
λ	Hari	λ	Hari	39	λ	80	λ

Figure 6

For example, α believes that object known as Tom to him is the same as the object known as Hari to λ and α chooses to borrow attribute values for his belief Tom from λ's belief of Hari. This is indicated by C_{salary} value being λ. However, α does not accept λ's belief of Ron and hence cannot borrow any attributes from Ron.

4. **Referential Integrity**: Let FK_1 be the foreign key of the referencing relation R_1 with key K_1. Let R_2 be the referenced relation with key K_2. An instance $r_1 \in R_1$ and $r_2 \in R_2$ satisfy referential integrity iff:
   a) $\forall t_{11} \in r_1, t_{11}[K_1] \neq$ null then $\exists t_{21} \in r_2$ such that $t_{11}[K_1] = t_{22}[K_2] \wedge t_{11}[BC] = t_{21}[BC] \wedge t_{11}[BC] \geq t_{21}[BC]$, and
   b) $\forall t_{11}, t_{12} \in r_1$ and $\forall t_{21}, t_{22} \in r_2$, if $t_{11}[K_1] = t_{12}[K_1] \wedge t_{11}[BC] = t_{21}[BC] \wedge t_{12}[BC] = t_{22}[BC] = t_{11}[BC] \wedge t_{11}[K_1] = t_{21}[K_2] = t_{22}[K_2] \Rightarrow t_{21}[K_1] = t_{22}[K_2]$.

3.5 SQL-like Statements:

In this section we present out SQL-like statements for SELECT, INSERT, UPDATE and DELETE.

3.5.1 The Select Statement

The select statement executed by a δ-user has the following general form[12]:

SELECT B_1 [, B_2] …
FROM R_1 [, R_2] …
[WHERE p]
[RESTRICTED TO q]

Symbol Explanation –
R_1, R_2, … are relation names.
B_1, B_2, … are data or classification attributes or any of the key values: OC, OO, BC, BO or wildcard (*).
p is a predicate expression that may include conditions involving data attributes, classification attributes, OC, OO, BC, BO values or a wildcard (*) that may concern all levels lower than or equal to δ. p can also include an embedded SELECT statement as in example 2.
q is a predicate expression used to specify the level(s) in the hierarchy that the selections should be restricted to (e.g. if a δ-user wants to specify 'search all beliefs at all levels below

[12] [ ] implies that the expression within [ ] is optional. This notation holds for all statements.

me that are not part of my belief' then the predicate q would be OC = * Λ OC ≠ 'δ'[13] and if a δ-user wants to specify 'search beliefs of all users below me that have been accepted by me' then the predicate q would be OC = 'δ' ∧ BC = *).

Example 1 Find all users who believe in the existence of or accept object John.

```
SELECT OC
FROM  R
WHERE  OO = 'John' V BO = 'John'
```

Example 2 Find object names believed by me (δ-user) but not by users at any level below me.

```
SELECT BO
FROM  R
WHERE  OO = BO Λ BO = * Λ BC NOT IN ( SELECT  BC
                                      FROM R
                                      WHERE OO = BO Λ BO = *
                                       RESTRICTED_TO OC = * Λ OC ≠ δ)
```

Select operation evaluates only tuples having t[OC] = t[BC] = δ unless there is RESTRICTED_TO clause. If there is more than one relation included in the FROM clause, the predicate p is implicitly substituted by p ∧ (R_1.OC = R_2. OC = …). For tuples t satisfying p, the data of t for the attributes listed in the SELECT clause are included in the result. A SELECT statement is assumed to always succeed, although the returned tuple may be an empty set.

Replacing p with p ∧ (R_1.OC = R_2. OC = …) serves to enforce that δ-tuples in one relation only join with δ-tuples in other relations. This is based on the idea that a δ-tuple contains all the data accepted by δ-users, and therefore should only be joined with other δ-tuples. Otherwise, it is difficult to interpret the returned results.

3.5.2 The Insert Statement

The insert statement executed by a δ-user has the following general form:

```
INSERT
INTO R ( OO, A_j1 [, A_j2] … )
VALUES (oo, a_j1 [, a_j2] …)
```

Symbol Explanation –
R is a relation name;
OO is the object belonging to user requesting insertion owner;
A_{j1}, A_{j2}, … are data attribute names in R;
oo, a_{j1} [, a_{j2}] … are the data values for OO, A_{j1} [, A_{j2}] …respectively satisfying relevant domain constraints. One or more a_{ji} could also be an embedded select statement in cases when data borrowing is needed[14].

[13] OC ≠ 'δ' can also be interpreted as ¬ OC = 'δ'

[14] If all a_{ji} values are embedded statements, this could imply that more than one tuple is being simultaneously inserted and this may cause a violation of the key-polyinstantiation integrity.

Each INSERT data manipulation can insert at-most one tuple into the relation R. The inserted tuple t is constructed as follows:
For $1 \leq i \leq n$,

1. The value OO (object name) must be specified in the attribute list of the INTO clause and value oo is in the attribute list of the VALUES clause otherwise data manipulation is rejected.
2. t[OO] = t[BO] = oo and t[OC] = t[BC] = 'δ' which means that the inserted tuple is believed by the δ-user.
3. If A_i is in the attribute list of the INTO clause and a_i is in the attribute list of the VALUES clause then, $t[A_i] = (a_i)$, else $t[A_i]$ = (null) which means that if the value of an attribute A_i is not specified in VALUES clause value of A_i is set to be null.
4. All C_i values are automatically set equal to 'δ' unless embedded SELECT statement specifying this value is used. If not, a δ-user can set a desired value for a C_i, classification attribute, using the UPDATE command after the tuple has been inserted.

Example 3 User μ wants to insert the new object Mary, borrowing the age value from object Hari in which λ believes.

```
INSERT
INTO R (*)
VALUES ('Mary', (SELECT R.AGE
                 FROM R
                 WHERE OC = λ Λ OO = 'Hari'), 80)
```

The resulting relation is[15]:

OC	OO	BC	BO	Age	C_{age}	Salary	C_{salary}
μ	Mary	μ	Mary	39	λ	80	μ
λ	Hari	λ	Hari	39	λ	47	λ

The insertion is permitted only if all integrity constraints are satisfied and there is no t' Є r such that t' [OC, OO, BC, BO] = t[δ, oo, δ, bo]. If so, the insertion tuple is rejected and data manipulation is rejected and the original database state is left unchanged. This rejection does not open a covert channel since a tuple causing this rejection is already visible to the δ-user.

When the tuple is inserted into the belief of the δ-user, it is automatically inserted into the beliefs of all users above δ such that for t'∈ all users above δ $\forall A_i, C_i$ $t[A_i, C_i] = t'[A_i, C_i]$ and value t'[OO] = null (until the higher user accepts this tuple by changing the OO value by using an UPDATE command).

3.5.3 The Update Statement

The update statement executed by a δ-user has the following general form:

UPDATE R
SET [OO = oo], [BO = bo] , $[A_{j1} = a_{j1}]$, $[C_{j1} = c_{j1}]$ [, $A_{j2} = a_{j2}$, $C_{j2} = c_{j2}$] …

[15] In all our examples we show the tuples that are directly related to the operation being illustrated in the example which is actually a subset of the actual belief of the user.

[WHERE p]

Symbol Explanation –
R is a relation name; *OO* is the owner object; *BO* is the belief object.
A_{j1}, A_{j2}, … are data attribute names in R; C_{j1}, C_{j2}, … are classification attribute names in R;
oo, bo, a_{j1} *[,* a_{j2}*]* … are the data values for OO, BO, $A_{j1,}$ C_{j1} [, $A_{j2,}$ C_{j2}] …respectively satisfying relevant domain constraints. Not all these pairs have to be present in SET clause.
p is a predicate expression that may include conditions involving data attributes, classification attributes, OO, BO, BC values or an embedded SQL statement.

Only those tuples t Є r that have t[OC] = δ are taken into consideration when UPDATE operation is evaluated. For tuples t Є r that satisfy the predicate p, r is updated as follows:
for $1 \leq i \leq n$,

1. t[OO] = oo
2. t[BO] = bo
3. $t[A_i] = a_i$
4. $t[Ci] = c_i$

The update is permitted only if all integrity constraints are satisfied and there is no t' Є r such that t' [OC, OO, BC, BO] = t[δ, oo, bc, bo]. If so, the tuple is rejected and data manipulation is rejected and the original database state is left unchanged.

Example 4 User μ accepts the object Jane that was inserted by λ.
UPDATE R
SET OO = 'Aaron'
WHERE OO = null Λ BC = λ Λ BO = 'Jane'

a)

OC	OO	BC	BO	Age	C_{age}	Salary	C_{salary}
μ	null	λ	Jane	39	λ	47	λ
λ	Jane	λ	Jane	39	λ	47	λ

b)

OC	OO	BC	BO	Age	C_{age}	Salary	C_{salary}
μ	Aaron	λ	Jane	39	λ	47	λ
λ	Jane	λ	Jane	39	λ	47	λ

Figure 7 a) relation before update b) the resulting relation after update

Example 5 User μ updates the object Jane that was inserted by λ.
UPDATE R
SET OO = 'Aaron', Salary = 80
WHERE OO = null Λ BC = λ Λ BO = 'Jane'

a)

OC	OO	BC	BO	Age	C_{age}	Salary	C_{salary}
μ	null	λ	Jane	39	λ	47	λ
λ	Jane	λ	Jane	39	λ	47	λ

b)

OC	OO	BC	BO	Age	C_{age}	Salary	C_{salary}
μ	Aaron	λ	Jane	39	λ	80	μ
λ	Jane	λ	Jane	39	λ	47	λ

Figure 8 a) relation before update b) the resulting relation after update

3.5.4 The Delete Statement

The delete statement executed by a δ-user has the following general form:

DELETE
FROM R
[WHERE p]

Symbol Explanation –
r is the relation name.
p is a predicate expression that may include conditions involving data attributes, classification attributes, OO, BO, BC values or an embedded statement.

Only those tuples t Є r that have t[OC] = δ are considered in the evaluation of DELETE operation, i.e. predicate expression is effectively changed to p Λ t[OC] = δ. For those tuples t Є r that are selected, r is changed as follows:

1. If t[OC,OO] = t[BC,BO] which indicates that the object to delete is a belief object and this object or its attributes might be borrowed by some upper level users. Then tuple can be deleted. And the next step would be to check if there exists a tuple t' belonging to a user δ' (δ' > δ) that borrowed elements from the tuple t, if so then in δ's belief, all corresponding Ci values should be set to null. In other words, $\exists t' \in r$ such that t'[BC,BO] ≠ t'[OC,OO] Λ t'[BC,BO] = t[OC,OO], then ∀t' t'[C_i] = null.
2. If t[OC,OO] ≠ t[BC,BO] which means that the object to delete was accepted by δ-user and neither this object or its attributes can be borrowed by other users by data-borrowing properties and requested relation can be safely deleted from the database.

Example 6 User μ does not accept the object Jane that was inserted by λ.

DELETE
FROM R
WHERE OO = null Λ BC = λ Λ BO = 'Jane'

a)

OC	OO	BC	BO	Age	C_{age}	Salary	C_{salary}
μ	null	λ	Jane	39	λ	47	λ
λ	Jane	λ	Jane	39	λ	47	λ

b)

OC	OO	BC	BO	Age	C_{age}	Salary	C_{salary}
λ	Jane	λ	Jane	39	λ	47	λ

Figure 9 a) relation before delete b) the resulting relation after delete

Example 7 User λ deletes the object Jane which belongs to him.

DELETE
FROM R
WHERE OO = BO Λ BO = 'Jane'

a)

OC	OO	BC	BO	Age	C_{age}	Salary	C_{salary}
μ	Aaron	λ	Jane	39	λ	80	μ
λ	Jane	λ	Jane	39	λ	47	λ

b)

OC	OO	BC	BO	Age	C_{age}	Salary	C_{salary}
μ	Aaron	λ	Jane	39	null	80	μ

Figure 10 a) relation before delete b) the resulting relation after delete

4. Performance Evaluation:

Since this is the first work that defines a MLS data model for a relational supporting key-polyinstantiation, we have no benchmark for a performance evaluation. Further, our effort till date has been concentrated on defining the model and the semantics for its manipulation; and without an implementation of the constructs, it is impossible to measure performance using metrics like performance time, throughput etc. Our focus has been on accuracy and representation and not on performance optimization.

5. Conclusion and Future Work:

In this paper, we defined a MLS data model with key-polyinstantiation that is sound, complete and free of downward information flows. The integrity properties - entity, null values, data borrow and referential integrity - strongly support the fact that our model is secure, unambiguous and that it is a powerful data model.

In our model, a subject can not only 'read down' i.e. get data accepted by itself and by subjects at levels below it; but also do some kind of 'write up'- i.e. change the data accepted by subjects at levels above it, provided that the data is owned by itself.

Our statements very closely resemble SQL and it is our belief that it should be possible to implement them in SQL with reasonable ease.

Our model also supports key-polyinstantiation which is essential for an accurate representation of the real world.

Future work would include semantics for the creation and modification of the user hierarchy lattice, creation, modification and deletion of relations and more advanced manipulation operations such as joins etc. Implementation of this model using SQL is another interesting open topic.

Another avenue for future work would be to extend the model to other databases - temporal, spatial, object-oriented etc. and within these frameworks to evaluate functional dependencies and their implications.

6. Acknowledgment

The authors thank Dr. Wallapak Tavanapong for her valuable comments.

7. References:

[CG1995] Tsz Shing Cheng and Shashi K. Gadia. An Algebra for belief persistence in multilevel security databases. Technical report, Department of Computer Science, Iowa State University, September 1995.

[GS1998] Shashi K. Gadia. Applicability of Temporal Data Models to Query Multilevel Security Databases: A Case Study. Technical report, Department of Computer Science, Iowa State University, 1998.

[JS1990] Tsz Shing Cheng and Shashi K. Gadia. Polyinstantiation integrity in multilevel relations. Proceedings of IEEE Symposium on Research in Security and Privacy, pp 104-115, 1990.

[JS1991] Sushil Jajodia and Ravi Sandhu. Toward a multilevel secure relational data model. Proceedings of ACM-SIGMOD, pp 50-59,1991.

[PMP1994] Niki Pissinou, Kia Makki, E.K. Park. Towards a Framework for Integrating Multilevel Secure Models and Temporal Data Models. Proceedings of ACM-SIGMOD, pp 280-287, 1994.

[SC1998] Ravi Sandhu and Fang Chen. The Multilevel Relational (MLR) Data Model. ACM Transactions on Information and System Security, Vol. 1, No. 1, November 1998, pp 93-132.

[SW1992] Ken Smith, Marianne Winslett. Entity Modeling in the MLS Relational Model. Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign, 1992.

[WSQ1994] Marianne Winslett, Kenneth smith, Xiaolei Qian. Formal Query Languages for Secure Relational Databases. ACM Transactions on Database Systems, Vol. 19, No. 4, December 1994, pp 626-662.